F3

Faculty of Electrical Engineering
Department of Computer Science

Master's Thesis

FIDO2 USB Security Key

Martin Endler
Open Informatics – Cybersecurity

May 2025
https://github.com/pokusew/fel-masters-thesis
Supervisor: Ing. Jan Sobotka, Ph.D.

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Endler Martin** Personal ID number: **483764**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Science**

Study program: **Open Informatics**

Specialisation: **Cyber Security**

II. Master’s thesis details

Master’s thesis title in English:

FIDO2 USB Security Key

Master’s thesis title in Czech:

FIDO2 USB bezpečnostní klíč

Name and workplace of master’s thesis supervisor:

Ing. Jan Sobotka, Ph.D. Department of Measurement FEE

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **02.09.2024** Deadline for master's thesis submission: **23.05.2025**

Assignment valid until: **15.02.2026**

Head of department’s signature

prof. Mgr. Petr Páta, Ph.D.
Vice-dean´s signature on behalf of the Dean

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work.
The student must produce his thesis without the assistance of others, with the exception of provided consultations.
Within the master’s thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt

Student’s signature

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Endler Martin** Personal ID number: **483764**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Science**

Study program: **Open Informatics**

Specialisation: **Cyber Security**

II. Master’s thesis details

Master’s thesis title in English:

FIDO2 USB Security Key

Master’s thesis title in Czech:

FIDO2 USB bezpečnostní klíč

Guidelines:

FIDO2 is a set of standards based on asymmetric cryptography that enables easy, secure, and phishing-resistant authentication.
The goal of this work is to create a new open-source implementation of a FIDO2 USB hardware external authenticator that is well-documented and thoroughly tested and offers a detailed yet accessible insight into the inner workings of FIDO2, which is something that existing implementations currently lack.
1. Make yourself familiar with the FIDO2 set of standards.
2. Review suitable technologies and existing similar projects.
3. Implement a working FIDO2 USB hardware external authenticator (“security key”) from scratch. External libraries can be used for some low-level generic components.
4. Follow software development best practices and use applicable software quality assurance methodologies.
5. Demonstrate the working of the implementation with authentication flows on real WebAuthn-enabled websites.
6. Document the work and make it publicly available on GitHub.

Bibliography / sources:

[1] W3C (April 8, 2021). Web Authentication: An API for accessing Public Key Credentials – Level 2. W3C Recommendation. https://www.w3.org/TR/webauthn-2/
[2] FIDO Alliance (June 21, 2022). Client to Authenticator Protocol (CTAP). CTAP 2.1 Proposed Standard. https://fidoalliance.org/specs/fido-v2.1-ps-20210615/fido-client-to-authenticator-protocol-v2.1-ps-errata-20220621.html
[3] USB Implementers Forum (April 27, 2000). USB 2.0 Specification. https://usb.org/document-library/usb-20-specification

FAKULTA ELEKTROTECHNICKÁ
FACULTY OF ELECTRICAL ENGINEERING
Technická 2
166 27 Praha 6

DECLARATION

I, the undersigned

Student's surname, given name(s): Endler Martin
Personal number: 483764
Programme name: Open Informatics

declare that I have elaborated the master's thesis entitled

FIDO2 USB Security Key

independently, and have cited all information sources used in accordance with the Methodological Instruction on the Observance of Ethical Principles in the Preparation of University Theses and with the Framework Rules for the Use of Artificial Intelligence at CTU for Academic and Pedagogical Purposes in Bachelor's and Continuing Master's Programmes.

I declare that I used artificial intelligence tools during the preparation and writing of this thesis. I verified the generated content. I hereby confirm that I am aware of the fact that I am fully responsible for the contents of the thesis.

In Prague on 21.05.2025

Bc. Martin Endler
.....
student's signature

Acknowledgement

First, I would like to thank my supervisor Ing. Jan Sobotka, Ph.D. for his guidance, support, and patience. Second, I would like to thank my family and close friends for always supporting me throughout my studies.

Abstract / Abstrakt

Passwords as a means of authentication suffer from many problems. To increase security, new technologies and standards have been introduced. FIDO2 is a set of standards that enables easy, secure, and phishing-resistant authentication with passwordless, second-factor, and multi-factor user experiences using embedded authenticators (such as device-embedded biometrics or PINs) or external authenticators (such as FIDO USB security keys). It is supported by major OSs, browsers, websites, and applications.

The goal of this project is to create a new open-source implementation of an external FIDO2 authenticator from scratch that would be well-documented, thoroughly tested, and production-ready. By focusing on the quality and the documentation, this implementation could help others understand the FIDO2 standards by providing a detailed yet accessible insight into the inner workings of these protocols (which the existing implementations lack). In general, this project has the potential to contribute to the popularization of FIDO2 technology.

Keywords: FIDO2, CTAP2, USB, WebAuthn, security key, passkeys, authenticator, passwordless authentication, multi-factor authentication, MFA, second-factor authentication, 2FA, USB, asymmetric cryptography, public-key cryptography

Passwords as a means of authentication suffer from many problems. To increase security, new technologies and standards have been introduced. FIDO2 is a set of standards that enables easy, secure, and phishing-resistant authentication with passwordless, second-factor, and multi-factor user experiences using embedded authenticators (such as device-embedded biometrics or PINs) or external authenticators (such as FIDO USB security keys). It is supported by major OSs, browsers, websites, and applications.

The goal of this project is to create a new open-source implementation of an external FIDO2 authenticator from scratch that would be well-documented, thoroughly tested, and production-ready. By focusing on the quality and the documentation, this implementation could help others understand the FIDO2 standards by providing a detailed yet accessible insight into the inner workings of these protocols (which the existing implementations lack). In general, this project has the potential to contribute to the popularization of FIDO2 technology.

Klíčová slova: FIDO2, CTAP2, USB, WebAuthn, bezpečnostní klíč, passkeys, autentizátor, přihlášení bez hesla, vícefaktorová autentizace, MFA, dvoufaktorová autentizace, 2FA, asymetrická kryptografie, kryptografie s veřejným klíčem

Překlad titulu:
FIDO2 USB bezpečnostní klíč

/ Contents

Chapter 1
Introduction

Passwords as a means of authentication suffer from many problems. To increase security, multi-factor and passwordless authentication have been introduced. **FIDO2** is a set of standards based on *asymmetric cryptography* that enables easy, secure, and phishing-resistant authentication. It supports *passwordless*, *second-factor*, and *multi-factor* user experiences with embedded (or bound) authenticators (such as biometrics or PINs) or external (or roaming) authenticators (such as FIDO Security Keys, mobile devices, wearables, etc.). All major OSs, browsers, and a growing number of websites and applications support FIDO2 (or the previous more limited standard FIDO U2F).

The goal of this project is to create a new open-source implementation of an external FIDO2 authenticator from scratch that would be well-documented, thoroughly tested, and production-ready. By focusing on the quality and the documentation, this implementation could help others understand the FIDO2 standards by providing a detailed yet accessible insight into the inner workings of these protocols (which the existing implementations lack). In general, this project has the potential to contribute to the popularization of FIDO2 technology.

The result of this project will be an open-source software implementation of a FIDO2 USB hardware external authenticator together with the final project report that will document it. The working of the implementation will be demonstrated on a suitable hardware platform (such as STM32F4). However, the hardware implementation is not the focus of this project.

Chapter 2
FIDO2

FIDO2 is a set of standards based on *asymmetric cryptography* that enables easy, secure, and phishing-resistant authentication. It supports *passwordless*, *second-factor*, and *multi-factor* user experiences with embedded (or bound) authenticators (such as biometrics or PINs) or external (or roaming) authenticators (such as FIDO Security Keys, mobile devices, wearables, etc.).

The specifications are:

- Client to Authenticator Protocol (CTAP) by FIDO Alliance
- Web Authentication (WebAuthn) API by World Wide Web Consortium (W3C)

Note: FIDO U2F is a predecessor of FIDO2 that can be used only for two-factor authentication (as a second factor).

2.1 Functional Description

The basic idea is that the credentials belong to the user and are managed by a WebAuthn/FIDO2 Authenticator, with which the WebAuthn Relying Party interacts through the client platform. Relying Party scripts can (with the user's consent) request the browser to create a new credential for future use by the Relying Party [1].

2.1.1 Relying party (RP)

An entity whose application utilizes the WebAuthn API to register and authenticate users, and which stores the public key.

2.1.2 Authenticator

A cryptographic entity that handles generating and storing keys and performing cryptographic operations.

The private keys never leave the authenticator.

2.1.2.1 Credential Storage Modality

The keys might be either stored in persistent storage embedded in the authenticator (necessary for client-discoverable/resident credentials that can be used as a first factor), or they might be derived from the credential ID (then the authenticator can support virtually an unlimited number of credentials).

For more information, see 6.2.2. Credential Storage Modality in [1].

2.1.3 Client

An entity that acts as an intermediary between the relying party and the authenticator (typically a web browser or a similar application).

2.1.4 Client Device

The hardware device on which the WebAuthn Client runs, for example a smartphone, a laptop computer or a desktop computer, and the operating system running on that hardware.

2.1.5 Client Platform

A client device and a client together make up a client platform. A single hardware device may be part of multiple distinct client platforms at different times by running different operating systems and/or clients.

Chapter 3
Existing Work

Before I started working on my own implementation, I did an extensive search to see what open-source implementations of FIDO2 authenticators were available.

3.1 Open-Source Software and Hardware

3.1.1 SoloKeys

- x

3.1.2 Nitrokey

- x

3.1.3 OpenSK by Google

- x

3.2 Other Relevant Projects

2 bachelor theses at CTU in Prague (both from CTU FIT):

- x

2 open-silicon designs:

- x

Chapter 4
Implementation

This chapter describes the main parts of our FIDO2 authenticator implementation.

Main points:

- Written in **C**.
- Runs on the STM3240G-EVAL board with the STM32F407IGH6 MCU.
- Uses STM32CubeF4 (HAL, LL, and USB Device Library) via STM32CubeMX generator.
- The complete source code is available on GitHub here: `https://github.com/pokusew/fel-krp-project`
- FIDO2 (specifically CTAP2) implementation in fido2 dir is based on the SoloKeys Solo 1 project. However, some parts of the original implementation have been rewritten / refactored / modified / fixed, so it could be used with the STM32F4 MCU (which has, for example, a different flash memory organization).

4.1 Architecture

In this section I describe individual layers (protocol stack) of the firmware that together implement the necessary FIDO2 functionality. The diagram on the right depicts these layers.

4.1.1 CTAP2

The core part is the implementation of the CTAP2 protocol. CTAP2 protocol is a high-level transaction-oriented protocol. A CTAP2 transaction consists of a request, followed by a response message. CTAP2 transactions are always initiated by the client. Messages are encoded using the concise binary encoding CBOR.

There are two essential transactions – authenticatorMakeCredential, which is implemented in ctap_make_credential, and authenticatorGetAssertion, which is implemented in ctap_get_assertion. They facilitate the register and authenticate flows as described in the Relying party (RP) section.

4.1.2 CTAPHID

Mapping of messages to the underlying USB transport is facilitated by the CTAPHID layer. Request and response messages are divided into individual fragments, known as packets. Packets are the smallest form of protocol data units, which in the case of CTAPHID are mapped into USB HID reports. CTAPHID also implements logical channel multiplexing so that multiple clients (browsers, apps, OS) on the same client device (device) can communicate with the authenticator concurrently.

4.1.3 USB

The USB HID and USB layers are implemented using the STM32Cube USB device library and its Custom HID class. It uses the USB_OTG_FS (USB 2.0 Full Speed) MCU peripheral. There are two endpoints (IN, OUT) with interrupt transfer (64-byte packet max, poll every 5 millisecond). The descriptors (device, config, interface, endpoints, HID report) are set up according to the CTAPHID specification (11.2.8. HID device implementation). The USB HID report descriptor plays a key role in the automatic device discovery. The CTAPHID protocol is designed with the objective of driver-less installation on all major host platforms. Browsers and other clients use the standard USB HID driver included within the OS. The HID Usage Page field within the authenticator's HID report descriptor is set to the value 0xF1D0, which is registered to the FIDO Alliance (see HID Usage Tables 1.5). The following screenshot shows the authenticator in the USB Device Tree Explorer in System Information on macOS 11.

The chosen Vendor ID (VID) and Product ID (VID) come from the pid.codes project. pid.codes is a registry of USB PID codes for open source hardware projects. They assign PIDs on any VID they own to any open-source hardware project needing one. For the initial version of this project, I used their testing VID/PID 0x1209/0x0001. In the future, when I have a more-production-ready authenticator, I might eventually ask for my own PID.

4.1.4 Persistent Storage

The most complex part is the implementation of state persistence. There are multiple pieces of data that need to be persisted. Namely, it is:

- the master key (used for storing non-discoverable credentials, see more info in Credential Storage Modality),
- the PIN (or the information that it is not has not been set yet) for user verification (UV) and invalid PIN attempts counter (note that the PIN is not stored in plaintext, instead it is stored as a salted hash),
- the global signature counter,
- and client-side discoverable credentials (formerly called resident credentials, the abbreviation `rk`, which is used in the code, stands for **r**esident **k**ey).

Note: This document uses the following notation: 1 KB (kilobyte) = 1024 B (bytes).

As a persistent storage, we used **a part of the 1024 KB of Flash memory** that is built into the STM32F407IGH6 MCU. By default, it is used for storing the program code (firmware). Because our firmware takes only about the first 104540 B (102 KB), i.e., around 10% of the total capacity, we can use the rest for our application data.

However, flash memory comes with **an important limitation** – one can only program (write) flash bits from 1 to 0. In order to write 1 in place of the already written 0 bits, one must **erase the whole memory sector** (the size of which can be in the range of KBs). This is generic to flash memory technology and is not specific to the STM32.

In case of STM32F407IGH6, its flash memory organization is as follows (also documented in Inc/flash.h):

A main memory block divided into 12 sectors (with different sizes, numbered 0-11):

- `4 sectors of 16 KB: sectors 0-3`
- `1 sector of 64 KB: sector 4`
- `7 sectors of 128 KB: sectors 5-11`

Our program occupies the sectors 0-4 (102 KB fits into the first 128 KB). So, we are only left with the big 128KB sectors for the application data. **Their erasure takes a significant amount of time** (seconds). Also, the total possible number of erasures is limited (**flash memory wear**). Thus, it is not possible to naively rewrite the whole sector when only a few bytes need to be changed (for example, when the global signature counter needs to be incremented).

There are special file systems designed to allow efficient use of such memories. However, for our use case, **we implemented a simple custom solution, adapted from the Solo 1 project** (where they faced the same challenge, but their MCU's sectors were only 1 KB big which still allowed for relatively fast erasures).

The implemented algorithms are conceptually types of **simple wear-leveling algorithms**. With our solution, the need for erasures is greatly reduced. Most of the time, all the changes in the authenticator state can be done without any erasure at all (so it takes no extra time).

Here are the relevant parts of the code that implement the persistence:

- flash memory utils in Inc/flash.h and Src/flash.c
- Inc/memory_layout.h that defines the used sector numbers
- ensure_flash_initialized() in Src/app.c for initialization of the memory layout upon the very first startup and after memory reset (using the app_delete_data debug command)
- persistence of the AuthenticatorState struct (contains the master key + PIN data)
  - authenticator_read_state()
  - authenticator_write_state()
- persistence of the global signature counter
  - ctap_atomic_count() in Src/device.c
  - Note: A special effort has been made to make the counter atomic. In case the flash write is interrupted (e.g., the authenticator is disconnected from the USB and the power is interrupted), the counter should keep its previous value. This is critical for the correct FIDO2 implementation because the counter must never be decremented.
- persistence of the client-discoverable credentials (resident credentials, resident keys)
  - ctap_reset_rk()
  - ctap_rk_size()
  - ctap_store_rk()
  - ctap_delete_rk()
  - ctap_load_rk()
  - ctap_overwrite_rk()

    This is the only part where the wear-leveling algorithm is not implemented yet. Fortunately, it has only a negligible impact, since the ctap_overwrite_rk is only called once when a new client-discoverable credential is being created. This is in contrast with ctap_atomic_count() and authenticator_read_state() that can be invoked even multiple times during a single transaction.

Chapter 5
Conclusion

In this project, **we successfully implemented** a working FIDO2 USB hardware external authenticator (also called FIDO2 USB security key).

First, we outlined the goal of the project and its motivation. Then, we established the necessary theoretical foundation by describing the Web Authentication and CTAP2 standards, which are together known as FIDO2. We also reviewed existing projects related to our goal.

Then, we proceeded to the actual implementation, which represents the main contribution of this project. We documented our decisions and the most important parts of the implementation. All code is versioned using Git and is publicly **available online on GitHub**[1]. The repository includes build and run instructions. There is also a **CI/CD pipeline** to ensure software quality. Our authenticator supports both client-discoverable credentials and an unlimited number of non-discoverable credentials. Thanks to the full support of client-discoverable credentials and user verification (UV) using PIN, it can be used as a first factor (passkey).

Finally, **we tested** our authenticator **with real websites** with different configurations (1st factor vs 2nd factor). In all cases, **it worked great**, including with Google Account and GitHub.

However, the authenticator is not production-ready. Even though we were able to make it work (with the help of the Solo 1 codebase), there are a lot of details and edge cases that are not correctly handled. The whole FIDO2/WebAuthn specification is very complex and a proper implementation would take much more time.

5.1 Future Work

There are a few more steps needed to achieve the ultimate goal of creating a new FIDO2-compliant implementation from scratch that would be thoroughly tested and production-ready.

Currently, our implementation uses a part of the Solo 1 project (the CTAP2 layer) that we need to replace with our own implementation.

Furthermore, we would like to use test-driven development and ensure there is sufficient test coverage. As of now, we have set up a CI/CD pipeline (so at least we can test that project builds correctly), but there are no unit/integration/e2e tests yet.

Ultimately, we would like to attempt to pass the FIDO Authenticator Level 1 (L1) certification[2].

[1] `https://github.com/pokusew/fel-krp-project`

[2] `https://fidoalliance.org/certification/authenticator-certification-levels/authenticator-level-1/`

References

[1] *W3C (April 8, 2021). Web Authentication: An API for accessing Public Key Credentials – Level 2. W3C Recommendation.* `https://www.w3.org/TR/webauthn-2/`.

Appendix A
Glossary

CBOR	■	Concise Binary Object Representation
CTAP	■	Client to Authenticator Protocol
FIDO	■	Fast IDentity Online Alliance
HTTP	■	Hypertext Transfer Protocol
IDE	■	Integrated Development Environment
JSON	■	JavaScript Object Notation
MCU	■	microcontroller unit
MFA	■	multi-factor authentication
OS	■	Operating System
USB	■	Universal Serial Bus
2FA	■	second-factor authentication